ROOT

Branches, Trees, and SEGFAULT

Getting Started

Instructions for obtaining and installing ROOT are available at: https://root.cern.ch/building-root

Download the file from github

Contains root file, makefile, class

Inside the ROOT File

```
$ root -l Example.root
[1] _file0->ls()
[2] tout->Print()
[3] tout->Show(45)
[4] tout->Draw(“gaus”)
[5] tout->Draw(“gaus:ts”)
[6] .q
```

.q exits ROOT, and if it’s not quitting fast enough, add more q’s

Looping Over Data

[1] tout->MakeClass(“name”)
Creates a name.C and a name.h file with all the branch names included
Try making one now

Analysis code will go in the C file
The .C files includes instructions on how to execute from the ROOT command line
Automatically runs on file it was generated from unless changed in the .h file

Process.cxx

In order to compile a main function is needed
This file provides that and creates a TChain to pass to the analysis code
A TChain behaves as TTree but all chained together

Compile the code, fix any bugs and then run
Use cp to duplicate the Example.root file and try to run over both with the TChain
Use a TBrowser to examine the results

Other ROOT Things to Know

- Pointers
- Display Options
- Python Capability
- Debugging

Pointers

Due to the design of ROOT, pointers are more natural to work with

Ie;

TH1F *h1=new TH1F(); vs TH1F h1 = new TH1F();

h1->GetXaxis()->SetTitle(“name”) vs h1.GetXaxis().SetTitle(“name”);

In general this shouldn’t matter though, and the use of pointers can lead to memory problems

In general though ROOT is designed around the idea of pointers

ROOT Display Options

One of the first things you should do with your analysis results is write another program to take the resulting plots and draw them in the way you want

Complied ROOT code won’t draw plots but you can save PDFs, and the TBrowser uses the Draw(“ALP”) options by default

By having a seperate drawing program you can edit and produce plots quickly without having to rerun your analysis code

ROOT Display Options

TH1(TH1F) is the default plot type in ROOT

TH2F produces a heat map

TGraph is for more traditional x,y plots

TGraphErrors if you want error bars

TStack allows you to stack multiple histograms (and set colors) on top of each other

Axes and Legends are all ROOT objects and can be created/modified similarly to histograms

ROOT-Python Interfacing

Or... “How to avoid writing ROOT code!”

Danika MacDonell

Why is root_numpy awesome?

- root_numpy allows you to easily convert data between a ROOT TTree file and a structured numpy array

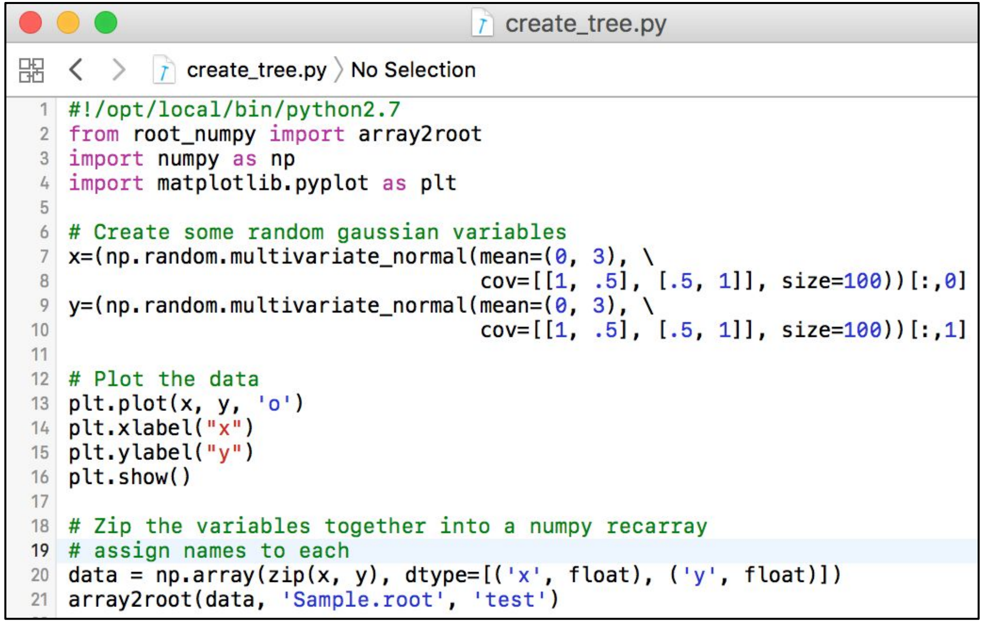
  The wonderful world of python is immediately at your fingertips!

- This can be done in as little as **1 line of code** (and very quickly!)

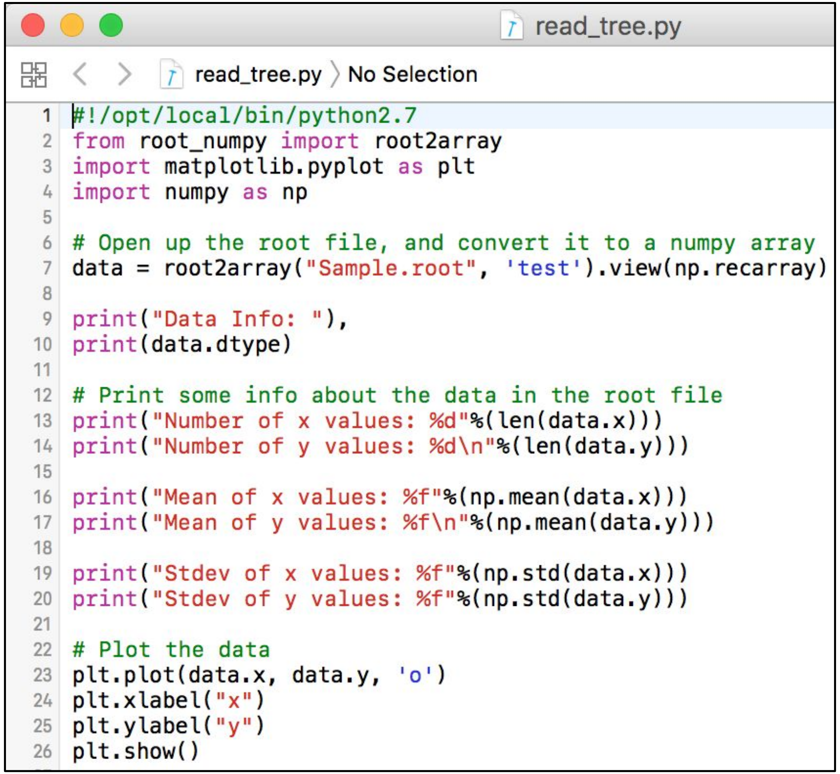
- When combined with the ROOT-Python interface package rootpy, root_numpy can also convert ROOT histogram files to/from numpy arrays

  ** Caveat: rootpy currently only supports python 2.7 (but this is being worked on!)

Create and read a ROOT TTree

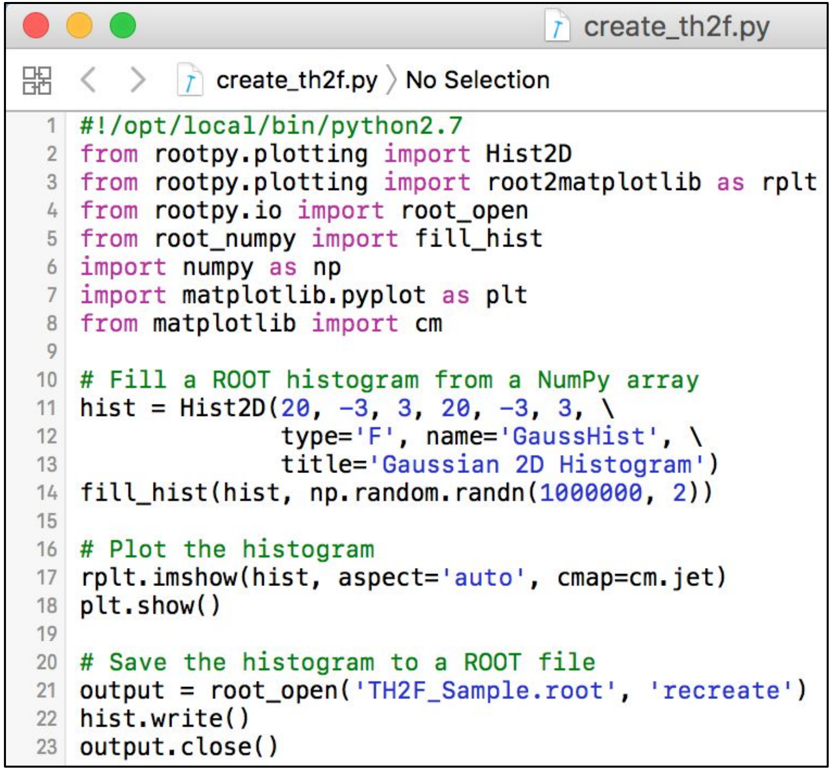


```python
#!/opt/local/bin/python2.7
from root_numpy import array2root
import numpy as np
import matplotlib.pyplot as plt

# Create some random gaussian variables
x=(np.random.multivariate_normal(mean=(0, 3), \
                                 cov=[[1, .5], [.5, 1]], size=100))[:,0]
y=(np.random.multivariate_normal(mean=(0, 3), \
                                 cov=[[1, .5], [.5, 1]], size=100))[:,1]

# Plot the data
plt.plot(x, y, 'o')
plt.xlabel("x")
plt.ylabel("y")
plt.show()

# Zip the variables together into a numpy recarray
# assign names to each
data = np.array(zip(x, y), dtype=[('x', float), ('y', float)])
array2root(data, 'Sample.root', 'test')
```

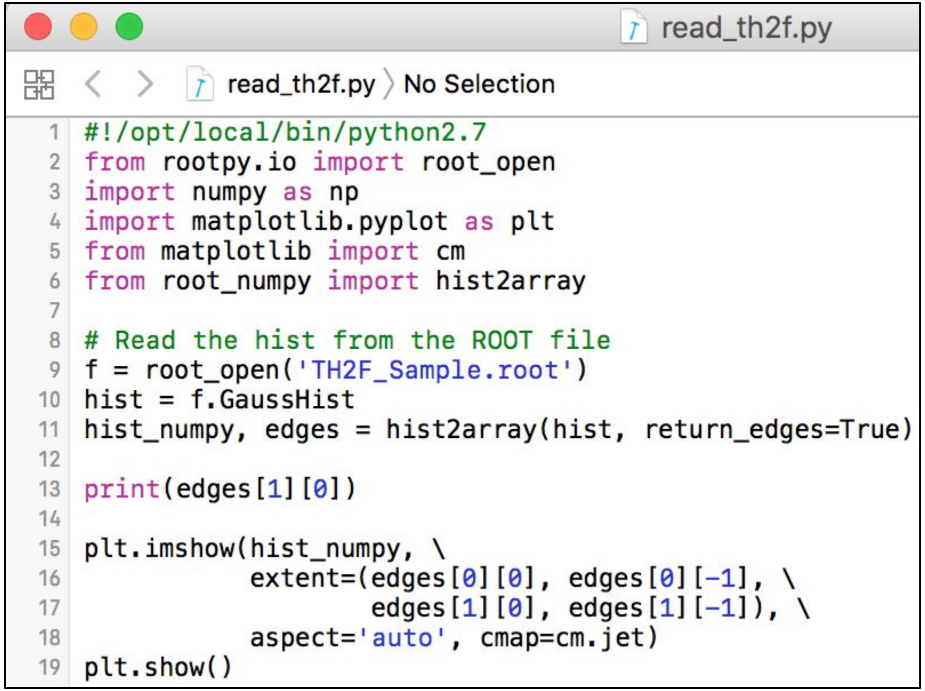


```python
#!/opt/local/bin/python2.7
from root_numpy import root2array
import matplotlib.pyplot as plt
import numpy as np

# Open up the root file, and convert it to a numpy array
data = root2array("Sample.root", 'test').view(np.recarray)

print("Data Info: "),
print(data.dtype)

# Print some info about the data in the root file
print("Number of x values: %d"%(len(data.x)))
print("Number of y values: %d\n"%(len(data.y)))

print("Mean of x values: %f"%(np.mean(data.x)))
print("Mean of y values: %f\n"%(np.mean(data.y)))

print("Stdev of x values: %f"%(np.std(data.x)))
print("Stdev of y values: %f"%(np.std(data.y)))

# Plot the data
plt.plot(data.x, data.y, 'o')
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

Create and read a ROOT TH2F (2D array of floats)



```
#!/opt/local/bin/python2.7
from rootpy.plotting import Hist2D
from rootpy.plotting import root2matplotlib as rplt
from rootpy.io import root_open
from root_numpy import fill_hist
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm

# Fill a ROOT histogram from a NumPy array
hist = Hist2D(20, -3, 3, 20, -3, 3, \
              type='F', name='GaussHist', \
              title='Gaussian 2D Histogram')
fill_hist(hist, np.random.randn(1000000, 2))

# Plot the histogram
rplt.imshow(hist, aspect='auto', cmap=cm.jet)
plt.show()

# Save the histogram to a ROOT file
output = root_open('TH2F_Sample.root', 'recreate')
hist.write()
output.close()
```



```
#!/opt/local/bin/python2.7
from rootpy.io import root_open
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from root_numpy import hist2array

# Read the hist from the ROOT file
f = root_open('TH2F_Sample.root')
hist = f.GaussHist
hist_numpy, edges = hist2array(hist, return_edges=True)

print(edges[1][0])

plt.imshow(hist_numpy, \
           extent=(edges[0][0], edges[0][-1], \
                   edges[1][0], edges[1][-1]), \
           aspect='auto', cmap=cm.jet)
plt.show()
```

Sample Installation Steps for ROOT-Python

For macports users:

1. >> sudo port install root6
2. >> sudo port install python27
3. >> sudo port install py27-pip
4. >> sudo pip-2.7 install numpy matplotlib root_numpy rootpy